METHOD AND APPARATUS FOR EXECUTING REMOTE PROCEDURES IN A REMOTE PROCESSOR FROM A CLIENT PROCESS EXECUTED IN A LOCAL PROCESSOR

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims benefit of U.S. Provisional Application No. 60/099,025 filed September 3, 1998 by Byron C. Darrah and Allen Klinger, and entitled "Procedure Gateway Interfacing," which application is hereby incorporated by reference herein.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to systems and methods for executing remote procedures, and in particular to a system and method for a method and apparatus for executing remote procedures in a remote processor from a client process executed in a local processor.

2. Description of the Related Art

Internet applications commonly need to access file and CPU resources of multiple computers. Current approaches to creating these programs involve special network server

software, and often, special protocols as well. This introduces avoidable overhead to software development and administration.

There are methods for implementing these programs by segmenting them into procedures that are distributed among different machines (as remote procedures). Systems which interface conventional remote procedures (including CORBA, RMI, RPC) come at the cost of administration and maintenance of the respective server software. For small applications this is "overkill". Even for large applications, the administrative access required can be expensive or unavailable.

Alternative methods typically involve the design of application-specific protocols and servers. Unfortunately, any resulting savings in administrative overhead is easily counterbalanced by the extra design effort.

What is needed is a simple enabling technology that allows developers to easily build applications that run on any networked computer and access the computing resources of Internet servers without the need to install any specialized remote procedure or protocol servers. The present invention satisfies that need.

SUMMARY OF THE INVENTION

To address the requirements described above, the present invention discloses a method, apparatus, and article of manufacture for executing remote procedures in a remote processor from a client process executed in a local processor.

The method comprises the steps of accepting a remote procedure call having information identifying the remote procedure including a global remote procedure locator; translating the remote procedure call into a CGI-compatible information transfer protocol; transmitting the translated remote procedure call to the remote processor interpreting the translated remote procedure call into a remote procedure-compatible format; and invoking the remote procedure in the remote processor. The article of manufacture comprises a data storage device tangibly embodying instructions to perform the method steps described above.

The apparatus comprises a client application program interface communicatively coupled to the server application program interface. The client application program interface translates a remote procedure call information identifying the remote procedure including a global remote procedure locator into a CGI-compatible information transfer protocol, transmits the translated remote procedure call to the remote processor, and interprets a remote procedure response into a client process-compatible format. The server application program interface interprets a remote procedure call translated by the client application program interface into a remote procedure-compatible format, invokes the remote procedure in the remote processor to produce a remote procedure output, and translates a remote procedure response into the CGI-compatible information transfer protocol.

One object of the present invention is to provide a vehicle for calling remote procedures without special server installations or special developer training.

Another object of the present invention is to provide a robust remote procedure service that is automatically restored following temporary server host downtimes.

Another object of the present invention is to provide a remote procedure gateway that allows for full functionality across firewalls.

Another object of the present invention is to provide a vehicle for calling remote procedures that is independent of the platform or programming language used at the client or the remote computer.

Another object of the present invention is to provide a remote procedure capability while protecting communications against port "hijacking" in multi-user systems.

The present invention provides a layer of communication and interfacing for program execution on multiple networked computers to support developing distributed procedures. This communication layer:

(1) Implements an operational layer of communication and interfacing for executing programs on multiple networked computers (by treating all hardware and software as if the system were a single computer, executing a coordinated set of routines);

(2) Requires minimal system administration overhead;

(3) Enables application program developers to create software for multiple network computers with minimal effort.

For example, a program using the present invention can run partly on a user's computer, and elsewhere on machines accessed over a network. In particular, some code would run on one or more remote web server hosts.

The present invention is applicable to a wide variety of distributed computing architectures, including those used for remote database access and Internet-enabled software such as educational and gaming software.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 is a block diagram showing an exemplary hardware environment for practicing the present invention;

FIG. 2 is a diagram illustrating an exemplary procedure gateway interface and related elements;

FIGs. 3A and 3B are flow charts presenting illustrative process steps used to practice one embodiment of the present invention;

FIGs. 4A and 4B are diagrams illustrating distinctions between the operation of the common gateway interface and the procedure gateway interface of the present invention;

FIG. 5 is a diagram illustrating an example of the use of a definition language to generate a client process to operate with a remote procedure; and

FIG. 6 is a diagram illustrating additional detail regarding the operation of the procedure gateway interface elements generated with a compiler.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

In the following description, reference is made to the accompanying drawings which form a part hereof, and in which is shown, by way of illustration, several embodiments of the present invention. It is understood that other embodiments may be

utilized and structural changes may be made without departing from the scope of the present invention.

I. HARDWARE ENVIRONMENT

FIG. 1 illustrates an exemplary computer system 100 that could be used to implement the present invention. The computer 102 comprises a processor 104 and a memory, such as random access memory (RAM) 106. The computer 102 is operatively coupled to a display 122, which presents images such as icons, windows, and interaction opportunities to the user on a graphical user interface 118B. The computer 102 may be coupled to other devices, such as a keyboard 114, a mouse device 116, a printer 128, etc. Of course, those skilled in the art will recognize that any combination or subset of the above components, or any number of different components, peripherals, and other devices, may be used with the computer 102.

Generally, the computer 102 operates under control of an operating system 108 stored in the memory 106, and interfaces with the user to accept inputs and commands and to present results using a graphical user interface (GUI) module 118A. Although the GUI module 118A is depicted as a separate module, the instructions performing the GUI functions can be resident or distributed in the operating system 108, the computer program 110, or implemented with special purpose memory and processors. The computer 102 also implements a compiler 112 which allows an application program 110 written in a programming language such as COBOL, C++, FORTRAN, JAVA, Perl, or other language to be translated into processor 104 readable code. After completion, the application 110 accesses and manipulates data stored in the memory 106 of the computer 102 using the relationships and logic that was generated using the compiler 112. The computer 102 also optionally comprises an external communication device such as a modem, satellite link, Ethernet card, infrared (IR) link, radio frequency (RF) transmitter, or other device for communicating with other computers or base stations, including cellular telephones, personal data assistants (PDAs), and other wireless networking devices.

In one embodiment, instructions implementing the operating system 108, the computer program 110, and the compiler 112 are tangibly embodied in a computer-readable medium, e.g., data storage device 120, which could include one or more fixed or removable data storage devices, such as a zip drive, floppy disc drive 124, hard drive, CD-ROM drive, tape drive, etc. Further, the operating system 108 and the computer program 110 are comprised of instructions which, when read and executed by the computer 102, causes the computer 102 to perform the steps necessary to implement and/or use the present invention. Computer program 110 and/or operating instructions may also be tangibly embodied in memory 106 and/or data communications devices 130, thereby making a computer program product or article of manufacture according to the invention. As such, the terms "article of manufacture" and "computer program product" as used herein are intended to encompass a computer program accessible from any computer readable device or medium.

Those skilled in the art will recognize that many modifications may be made to this configuration without departing from the scope of the present invention. For example, those skilled in the art will recognize that many combinations or subsets of the above components, or any number of different components, peripherals, and other devices, may be used with the present invention.

II. SUMMARY

The Procedure Gateway Interface (PGI) provided by the present invention is a system for executing remote procedures using a Common Gateway Interface (CGI)-compatible protocol (such as HTTP 1.0) as the underlying network protocol for remote procedure requests and transfers resulting data. The Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers, such as HTTP or web servers. (The term “web,” as used herein, refers to a system of geographically distributed or dispersed computers connected by communication links, such as the world wide web). A plain HTML document that a web daemon retrieves is static, which means it exists in a constant state: a text file that doesn't change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information.

PGI is programming language-independent and computer platform-independent. In contrast to other remote procedure protocols, PGI is intended to be very simple and require practically no server administration overhead. Remote PGI procedures require only a web server with a protocol such as HTTP 1.0 and CGI support. Therefore, nearly every Internet web server already fully supports PGI.

FIG. 2 is a diagram illustrating an exemplary PGI and related elements. PGI includes a client application program interface 206 and a remote procedure application program interface 212. The client application program interface (API) 206 translates remote procedure calls from a client process 204 running on a local or client computer 218 into a CGI-compatible information transfer protocol, such as HTTP 1.0. The remote procedure may include a remote procedure input value, input attributes, and information identifying the remote procedure. In one embodiment, this information includes a global remote procedure locator such as a universal resource locator (URL). The client API 206 also transmits translated remote procedure calls to a remote processor 220 via a server such as the web server 208 and server API 212.

The server API 212 interprets the remote procedure call that was translated by the client API 206 into a remote procedure compatible format, invokes the remote procedure in the remote processor 220. The remote procedure executes, and using the remote procedure input value (if provided), and returns remote procedure response that may include a remote procedure output value that was retrieved, for example, from a database file 216. The server API 212 then translates the remote procedure response into a CGI compatible information transfer protocol such as HTTP, and transmits the translated remote procedure response to the client API 206 in the client processor 218 via the server 208. The client API interprets the remote procedure response into a client process-compatible format, and returns the parameters to the client process 204.

FIGs. 3A and 3B are flow charts presenting illustrative process steps used to practice one embodiment of the present invention. A remote procedure call is accepted 302 by the client API. The remote procedure call is translated 304 into a CGI-compatible information transfer protocol, such as HTTP. The translated remote procedure call is then transmitted 306 to a server API 212 in remote processor 220. The server API 212

interprets 308 the translated remote procedure call into a remote procedure-compatible format, and invokes 310 the remote procedure in the remote processor to produce a remote procedure response, which may include a remote procedure output value.

As shown in FIG. 3B, the server API 212 translates 312 the remote procedure response into a CGI-compatible information transfer protocol, and transmits 314 the translated remote procedure response to the client API 206 in the local processor 218. The client API 206 interprets 316 the remote procedure response into a client process-compatible format, and provides 318 the translated remote procedure response to the client process 204.

As demonstrated by the foregoing, PGI uses APIs 206, 212 that handle the details of network communication with remote procedures 214 (e.g., passing of parameters back and forth). In one embodiment, the method uses HTTP and CGI protocols communicating with, and executing code on, remote web server 208 hosts.

A remote procedure 214 invoked via PGI behaves like a conventional local remote procedure except that it invokes its API twice. The first API invocation initializes the activity by forwarding required input parameters. When computation has ended, the second API invocation returns any output parameters. Then the remote procedure 214 exits.

III. APPLICATION PROGRAM INTERFACES

PGI is capable of extending support to any computer programming language, as long as there is a way for programs written in the language to communicate via HTTP. The PGI software includes an Application Programming Interface (API) for each supported programming language. The API handles the work of marshalling procedure parameters and using the PGI protocol to send them over an HTTP connection.

The API has two distinct parts: An interface for programs that *call* remote procedures, and an interface for procedures that *are called* remotely. The former is referred to as the *client* API 206, while the latter is referred to as the *agent* API 212 (we refrain from using the word "server" in order to avoid confusing it with HTTP servers).

Procedures are not restricted to exclusively be either clients or agents. A single procedure may act as both by being invoked by some client and making calls to other agents.

In PGI, the API for each supported computer language is implemented according to the conventions of the language. For example, the JAVA interface provides methods in a class library. A "C" interface provides header files and linkable object files or libraries, while a "Perl" interface provide a Perl package, and so on. When support for a new programming language is added to the API, it must obey the conventions of that language, but should otherwise be similar to an existing PGI language interface where possible. For example, functions, procedures, methods, subroutines, etc. which have the same functionality should have the same name.

IV. PGI PROTOCOL

Each PGI remote procedure 214 is stored in a separate executable file. The executable is treated like a CGI program in that it is installed so it can be executed by requesting an appropriate URL through a web server 208. From the perspective of the server 208, PGI remote procedures may be regarded as normal CGI programs. PGI, however, differs from ordinary CGI programs.

FIGs. 4A and 4B are diagrams illustrating distinctions between ordinary CGI and PGI. FIG. 4A shows that CGI itself can be used to let users start a program on a remote computer. It can also display the output of the program to the user. However, with CGI, the entire program must reside on the remote computer. Any computing resources such as printers or disk storage on the user's computer are inaccessible to the program. FIG. 4B illustrates a PGI program, which can execute code on both the user's local machine and the remote server host as needed.

A. Communication Encoding

The following defines the way PGI clients and remote procedures communicate through the PGI APIs. In one embodiment, PGI client processes 204 and remote procedures 214 communicate by sending streams of ASCII characters. Since virtually any data can be sent as a stream of ASCII characters, this does not limit the kind of text or any

other data that can be exchanged between clients processes 218 and remote procedures 214. This is simply the medium chosen to represent that data while it is being communicated across the network. In the following description, strings of characters are used to represent communications where the left-to-right order is the same as the first-to-last order, in time, of transmission.

The characters themselves are 8-bit bytes with unsigned integer values that map to ASCII digits, letters, and other symbols. Ordinarily, low-level details such as the ordering of bits within a byte is handled by network protocols at lower levels than PGI.

B. PGI Data Encoding

Whenever data values are communicated between PGI clients processes 204 and remote procedures 214, they are encoded according to their data type:

INTEGER: String of decimal digits with an optional '-' prefix.

BYTE STRING: The string of bytes. However, any bytes which do not have ASCII mappings or which represent ASCII control characters must be specially encoded as a '%' character followed by two hexadecimal digits representing the value of the byte (The characters 'abcdef' may be used as hexadecimal digits). Therefore, byte values 00-1f and 80-ff (hex) must be so encoded. In addition, the characters '=', '&', and '<' must also be encoded in this manner.

FLOATING POINT: Optional '-' followed by string of digits, followed by optional '.' and a string of digits followed by optional 'e+' or 'e-' and a string of digits, interpreted in decimal.

C. Invoking a Remote Procedure

When invoking a remote procedure 214 using a uniform resource locator and HTTP as the information transfer protocol, the client computer 218: (1) specifies the URL corresponding to the remote procedure; (2) specifies the name of the remote procedure and the argument signature; (3) establishes an HTTP connection to the agent's host (4) uses an HTTP POST request to activate the agent, and passes any input parameters to the agent. In this context, the agent is software executing on the computer being called to

execute a remote procedure. The agent's host is that computer that is executing the remote procedure (i.e. remote processor 220 in FIG. 2). In one embodiment, inputs are provided using CGI name/value pairs as follows:

ARG1 = {value-1} & ARG2 = {value-2} &...& ARGn = {value-3}

where each value-i is a PGI data-encoded value.

In one embodiment, only a remote procedure's input parameters are passed. For example, if a remote procedure 214 has five parameters, but two of them are output parameters, then the client will send ARG1= {value1} ... ARG3 = {value-3}, etc. The client then waits for results to be returned from the agent.

D. Returning Remote Procedure Results to the Client

After parsing the input and executing, the agent API 212, acting as a CGI program, returns a document of type application/octet-stream to the client which can contain one of the following two kinds of transmissions:

<PGI HELLO>ARG1={value-1}&ARG2={value-2}&...&ARGn={value-3}<PGI END>

or:

<PGI ERROR>{text}<PGI END>

Where the values and text are PGI-encoded. In one embodiment, only the remote procedure's output values are sent. There may be other data before and after this portion of the transmission, but such information is ordinarily ignored by the client. This PGI protocol may be extended to add more information about execution status and results to the transmission.

V. PGI DEFINITION LANGUAGE (PDL)

Using PGI, a developer writes one main program (the client process 204) and program files containing procedures to be executed on remote server hosts. The main program uses PGI to invoke a remote procedure.

In one embodiment, the software developer writes such programs with the use of a PDL compiler 210, which accepts information to specify remote procedures in a PGI Definition Language (PDL). The PDL compiler 210 reads PDL specifications created by

a program developer and generates source files that handle network communication and other aspects of remote procedure calling. The PDL compiler 210 supports generating these source files for a variety of programming languages, including "C", JAVA or Perl.

A. PDL Syntax and Definitions

There are only three kinds of statements in PDL: A comment, a server URL, and a procedure declaration. A comment is a line of text that begins with a '#' character and may contain any arbitrary text. The formal syntax for the other kinds of statements is shown below.

```
SERVER_URL:     server_url " URL " ;
PROCEDURE:      procedure WORD ( PARMLIST ) ;
      |         procedure WORD ( )
PARMLIST:  PARAMETER
      |         PARMLIST , PARAMETER
PARAMETER:      MODE TYPE WORD
MODE:           in
      |         out
TYPE:           int
      |         double
      |         string
```

In this syntax definition, URL indicates a string of text forming a valid URL (RFC 1736) except that the resource contains one '*' character. WORD indicates a string of letters, numbers, and underscores. Literal keywords are shown in lower case, although PDL is not case-sensitive. A PDL file may contain any number of SERVER_URL statements and PROCEDURE statements.

Each PROCEDURE statement gives the name of a remote procedure along with its argument signature. For example, the procedure statement "procedure Lookup(in string Name, out string Address);" declares a procedure named "Lookup" which takes a single input string called Name, and returns a single output string called "Address".

Each SERVER_URL statement specifies a default URL for the following procedures. When a default URL is used by a client, the '*' character is replaced by the name of the remote procedure being called. For example: "Server_URL "http://myhost.com/cgi-bin/*.cgi" indicates a URL for remote procedures on a machine called "myhost.com". Each SERVER_URL statement supersedes any previous ones.

B. Compiler Output

The PDL compiler 210 is capable of generating output for a variety of target languages. The exact output produced by the compiler is therefore dependent on the selected output language. However, regardless of the selected language, a general scheme is followed.

For each remote procedure, the compiler 210 produces a "shell" source file. The shell file includes all the code needed to communicate with PGI clients, invoke the procedure with appropriate input parameters as received over the network, and send the procedure's outputs back to the client over the network.

When a developer builds remote procedures, the shell file is used together with the developer's own program code to produce an executable file which can be treated as a CGI program.

For the client, the compiler creates a "stubs" file or package. The stubs file includes all the code needed to invoke PGI remote procedures, send appropriate input parameters over the network, and retrieve the procedure's outputs.

From the developer's perspective, the shells and stubs generated by the PDL compiler appear to handle the details of remote procedure calls transparently. The developer may write programs as though all procedure calls were simply local ones, knowing that the stubs and shells will introduce remote calling functionality.

Writing programs that use PGI very similar to writing conventional programs that do not use remote procedure calls. There are only a few small differences. First, PGI does not maintain any state or global environment. Hence, remote procedures may not reference global variables. A remote procedure must receive all of the data that it needs through its input parameters. Second, remote procedures must use input and output

parameters that have data types compatible with PGI. For example, in C, a developer can not pass a value of type time_t without first converting it to an integer or string that PGI supports. Third, the developer must create a short PDL file to declare any remote procedures.

There is one optional consideration as well. A client may specify a server URL for a remote procedure at run time, overriding any default that may have been given in the PDL specification. This is accomplished by calling the "Server_URL" function for a particular remote procedure. The semantics for calling this function vary slightly depending on the target language, but the output from the PDL compiler will give an exact specification.

Once a program with remote procedures has been created, the remote procedures are installed as CGI programs for a web server. When called, the remote procedure will then execute on the server host with privileges and access equal to that of any similarly installed CGI program.

FIG. 5 is a diagram illustrating an example of the use of PDL in enabling a main program to operate with one remote procedure, both of which are written in the "C" programming language. The main program 502, and procedure 504, are created by the user. The user also creates a declaration of the procedure in PDL 506. The PDL compiler 210 uses the declaration to generate code in the desired programming language ("C" in the illustrated example). The PDL compiler 210 produces client stubs 508, a procedure shell 510, and procedure prototypes 512 all for the target language, "C". These code elements work together with the user's code to enable the remote procedure 504 to be called remotely by the main program 502. The procedure shell 510, prototypes 512, and user's procedure code 504 are compiled by compilers 514 and 516 and the resulting object code 520 and 524 is linked using linkers 524 and 526 with the "C" PGI API 518 to produce the remote procedure 214. The client stubs 508, prototypes 512, and user's main program code 502 are compiled to produce the client program 218.

FIG. 6 is a diagram illustrating additional detail regarding the operation of the PGI elements such as those generated with the PDL compiler 210. Program code written by the software developer initiates a call to the remote procedure 214. Because the

procedure is remote, the PDL stub 508 passes each of the inputs to the client application through API 206, and encodes the inputs 604 to prepare them for transmission across the network. After the inputs are encoded, the PDL stub 508 passes them to the client application PGI 206 again, from where they are sent 606 to the remote procedure 214. The client application API 206 connects to a waiting web server 208 to execute a CGI program, and sends the encoded inputs across the (e.g. Internet) connection. The server API 212 gets 610 the inputs, decodes them 612, and provides them to the remote procedure via the shell 510. The shell 614 also invokes the remote procedure 214, and passes the inputs to it.

The remote procedure then executes, and returns any outputs to the shell 510. The outputs are then encoded 616, and the PDL 510 shell passes the encoded outputs to the server API 212 so they can be sent across the network back to the client process 204. The server API 212 sends the outputs to the web server 208, which handles them as an output from a CGI program, and sends the encoded outputs across the network connection. The client API 206 receives the encoded outputs and returns execution control to the stub 508. The stub 508 then collects the outputs from the client API 206 and uses the client API to decode the outputs. Finally, the outputs from the remote procedure are returned to the client process 204.

VI. EXAMPLES

A. Exemplary JAVA Implementations

The coding details of how the PGI API is used to set up input and output parameters depend on the computer language used. For example, PGI can be implemented in the JAVA programming language. A JAVA implemented API provides methods to the client and agent for giving and reading parameters, specifying a proxy server, and invoking remote procedures. When parameters are passed between a client and agent, they must be read in the same order that they are given. The following example presents a JAVA version of the PGI API and demonstrates its use.

1. JAVA Client API

The JAVA client API provides the following methods for use by client programs:

public RemotePGIProcedure(String url_str): a constructor for instantiating a remote procedure. The "url_str" parameter specifies the remote procedure to be called.

public static void setProxy(String url_str): specifies an HTTP proxy server to be used.

public void arg(int a): specifies an integer value as an input argument to a remote procedure.

public void arg(String a): specifies a string value as an input argument to a remote procedure.

public void arg(double a): specifies a floating point value as an input argument to a remote procedure.

public void call() throws PGIException: invokes a remote procedure. Any errors will result in a PGIException being thrown.

public int getInt(): retrieves an integer output parameter from a remote procedure that was called.

public String getString(): retrieves a string output parameter from a remote procedure that was called.

public double getDouble(): retrieves a floating point number output parameter from a remote procedure that was called.

A JAVA PGI client must: (1) import the PGI API; (2) instantiate an object for each remote procedure to be called; (3) specify input arguments to remote procedures; (4) call remote procedures; (5) retrieve any output arguments from remote procedures that are needed.

2. JAVA Agent API

The JAVA agent API provides the following methods for use by agent procedures:

public static int getInt(): gets an integer value input argument

public static String getString(): gets a string value input argument.

public static double getDouble(): gets a floating point value input argument.

public static void output(int a): specifies the value of an integer output parameter.

public static void output(String a): specifies the value of a string output parameter.

public static void output(double a): specifies the value of a floating point output parameter.

public static void pgiReturn(): returns from a remote procedure that was called by a client.

The JAVA agent remote procedure must: (1) import the PGI API; (2) read any needed input parameters; (3) specify values for its output parameters; (4) call pgiReturn immediately before it exits.

An option allows the class containing the agent procedure to extend the PGIProcedure class, but this is not necessary since all of the interface methods are public. The PGIProcedure class may be instantiated, but this is also not necessary since all of the interface methods are static.

3. JAVA Client Process

The following program shows a JAVA client program, "MyClient," which calls a remote procedure named "MyAgent." "MyAgent" takes one integer input parameter and returns one integer.

```
// MyClient uses the PGI API to make a call.
import pgi.*;
public class MyClient
{
try
{
int i = 5;
RemotePGIProcedure MyAgent = new
RemotePGIProcedure("http://www.cs.ucla.edu/~darrah/cgi-bin/MyAgent.cgi");
MyAgent.arg(i);
```

```
MyAgent.call();
i = MyAgent.getInt();
}
catch (Exception e) {System.out.println(e.toString()); }
}
```

4. Remote Procedure

The following example program shows the agent procedure, MyAgent, which is called by the client MyClient from the previous example. It takes an integer parameter, multiplies it by two, and returns the result.

```
// MyAgent is executed via PGI.
import pgi.*;
public class MyAgent extends PGIProcedure
{
public static void main(String[] args)
{
int i = getInt();
i *= 2;
output(i);
pgiReturn();
}
}
```

The foregoing PGI agent procedures need only to be built and installed as Common Gateway Interface (CGI) programs on a computer that hosts an HTTP server. The exact details for installing a CGI program vary slightly with developer preferences and server policies. However, each agent will typically be configured as a stand-alone executable program file and either placed in a special directory for CGI programs, or else named with a ".cgi" suffix so that the HTTP server can recognize the file as a CGI program.

Thus, since it is a CGI program, each agent has some associated URL. A PGI client program must use these URLs to specify agents to be invoked through the PGI API.

B. Exemplary "C" Implementation

Presented hereinafter is an example showing application of the PDL compiler 210 to automatically integrate the PGI API with the software developer's code (written in "C"). In this example, the software developer need not code and use the API directly.

1. PDL

A sample PDL file to be used by the PDL compiler in the "C" programming language is shown below:

```
# PGI Interface Definition for the TestProc procedure
#
# This file is used by the PDL compiler. If the target
# language is C, then the compiler will create the
# following files:
#
# ClientStubs.c, PGI_Procedures.h, and TestProcShell.c
# Tell the compiler what default server URL to use for
# the following remote procedures. It may be overridden
# at run time by using:
# TestProc_Server_URL(char *new_url);
Server_URL "http://www.clamso.net/cgi-bin/*";
Procedure TestProc(  in int a, in string b,
in double c, out int x,
out string y, out double z);
# End of file.
```

2. Client process

An exemplary embodiment of a client process authored in the "C" programming language is presented below:

```
#include <stdlib.h>
#include <stdio.h>
#include "PGI_Procedures.h"
extern int pgi_error;
extern char *pgi_error_str;
int main(void)
{
        /* Here are some things to pass to a remote procedure. */
        int a = 5; char *b = "Hello"; double c = 0.5;
        /* We'll use these variables to store the results. */
        int x; char *y; double z;
        /* Call the remote procedure. */
        TestProc(a, b, c, &x, &y, &z);
        /* Optional -- Handle errors */
        if(pgi_error)
        {
                printf("PGI Error: %s\n", pgi_error_str);
                return(0);
        }
        /* All done. Now print the results. */
        printf("x = %d\n", x);
        printf("y = %s\n", y);
        printf("z = %f\n", z);
        free(y);
        return(0);
}
```

3. Remote Procedure

Below is an exemplary embodiment of a remote procedure authored in the "C" programming language:

```
#include "PGI_Procedures.h"
void TestProc(int a, char *b, double c,
int *x, char **y, double *z)
{
        /* Just do some simple transformations to the inputs. */
        *x = a*2;
        *y = (char *)malloc(strlen(b)+7);
        sprintf(*y, ">>>%s<<<", b);
        *z = c/2.0;
}
```

Conclusion

This concludes the description of the preferred embodiments of the present invention. As the foregoing demonstrates, the present invention provides numerous advantages over prior art systems for interfacing remote procedures (CORBA, RMI, RPC, etc.) and custom-design approaches. Namely, the present invention provides a high degree of simplicity for writing programs to access remote resources, (2) does not require a "registry" database of procedures or methods, thus allowing for reduced administrative overhead, (3) The present invention does not require a specialized remote procedure or method server, (4) provides automatic functionality restoration following temporary server host downtimes (as may be experienced during electrical power outages), (5) provides more secure data exchange on servers with multi-user operating systems (such as UNIX) (this is accomplished by reserving some TCP/IP port, such as port 80, for data interchange), and (6) provides full functionality across firewalls, requiring only the existence of an HTTP proxy server. Further, these benefits and advantages can be realized without requiring action on the part of a system administrator with special access privileges (e.g. for modifying server start up configurations, installing special server software, and granting access to software developers).

The foregoing description of the preferred embodiment of the invention has been presented for the purposes of illustration and description. It is not intended to be

exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto. The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.